Un cadre théorique pour la gestion de grandes bases de motifs[1]

François Jacquenet, Baptiste Jeudy et Christine Largeron

Laboratoire Hubert Curien, UMR CNRS 5516, St-Etienne
prenom.nom@univ-st-etienne.fr

Résumé. Les algorithmes de fouille de données sont maintenant capables de traiter de grands volumes de données mais les utilisateurs sont souvent submergés par la quantité de motifs générés. En outre, dans certains cas, que ce soit pour des raisons de confidentialité ou de coûts, les utilisateurs peuvent ne pas avoir accès directement aux données et ne disposer que des motifs. Les utilisateurs n'ont plus alors la possibilité d'approfondir à partir des données initiales le processus de fouille de façon à extraire des motifs plus spécifiques. Pour remédier à cette situation, une solution consiste à gérer les motifs. Ainsi, dans cet article, nous présentons un cadre théorique permettant à un utilisateur de manipuler, en post-traitement, une collection de motifs préalablement extraite. Nous proposons de représenter la collection sous la forme d'un graphe qu'un utilisateur pourra ensuite exploiter à l'aide d'opérateurs algébriques pour y retrouver des motifs ou en chercher de nouveaux.

1 Introduction

La quantité d'information stockée dans les bases de données du monde entier n'a cessé de croître au cours du temps. Afin d'explorer ces mines potentielles de connaissance, des outils de fouille de données ont été conçus depuis plusieurs années. Ainsi, il est maintenant possible de fouiller de très grandes bases de données afin d'en extraire toute sorte de motifs modélisant de la connaissance. Selon les outils utilisés par les utilisateurs finaux pour leurs besoins, les motifs extraits peuvent être très variés. On peut citer par exemple les arbres de décision, les règles d'association, les concepts formels, etc. Toutefois, alors que la fouille de très grandes bases de données est devenue une tâche relativement aisée pour les utilisateurs finaux, ceux-ci sont maintenant confrontés à un nouveau problème : comment vont-ils pouvoir exploiter les grandes masses de motifs couramment extraites par les outils de fouille de données ? En fait, de la même façon qu'il était impossible il y a quelques années d'extraire manuellement de la connaissance à partir de grandes bases de données, il est de nos jours impossible de gérer de très grands volumes de motifs et les utilisateurs finaux sont donc en attente de nouveaux outils pour résoudre ce problème.

[1]Ce travail a été partiellement soutenu par le projet BINGO de l'ACI Masse de données 2004 - 2007, financé par le ministère de la recherche.

Deux approches ont été proposées afin de gérer et explorer ce qu'il est maintenant courant d'appeler des *bases de motifs*. La première est basée sur le concept de bases de données inductives (Imielinski et Mannila (1996); Boulicaut et al. (1999); Raedt et al. (2002); Raedt (2002)). En Europe, le projet CInQ[2] a joué un rôle majeur pour le développement des recherches dans ce domaine. Une base de données inductive contient non seulement des données mais également des motifs. De plus, des langages de fouille de données intégrés dans les systèmes de gestion de bases de données inductives offrent des facilités pour manipuler les motifs au travers d'opérateurs de post-traitement (Boulicaut et Masson (2005)). Toutefois, ceux-ci sont très rudimentaires et des systèmes de gestion de bases de motifs devraient fournir des fonctionnalités bien plus puissantes.

La seconde approche pour gérer des motifs s'intéresse justement à la notion de *système de gestion de bases de motifs* (SGBM). Dans Catania et al. (2004); Catania et Maddalena (2006), un SGBM est défini comme *un système permettant de supporter (stocker, traiter, rechercher) des motifs extraits à partir de données brutes afin de proposer des techniques efficaces de mise en correspondance de motifs et d'exploiter des opérations sur les motifs générant ainsi de l'information sous forme intentionnelle*. En fait, le principe consiste à stocker des motifs extraits par des outils de fouille de données en utilisant des structures de données efficaces. Des langages de manipulation de motifs doivent alors ensuite être conçus afin de les gérer. Cette approche induit deux questions. La première concerne la possibilité de concevoir un modèle générique pour représenter les motifs, la seconde concerne le langage nécessaire pour accéder aux motifs et les interroger. Le projet Panda[3] est un travail intéressant dans ce sens. Il propose un cadre générique permettant de modéliser diverses classes de motifs, puis un certain nombre d'opérateurs de type SQL permettant à l'utilisateur de les gérer. Cependant, comme le modèle sous-jacent pour stocker les motifs est le modèle relationnel, les requêtes conçues par les utilisateurs sont souvent très complexes, non intuitives et très consommatrices de temps de calcul. Même si SQL peut être considéré comme un candidat évident pour gérer des collections de motifs, ce langage a été en fait conçu pour accéder à des données stockées dans des bases de données et n'est donc pas bien adapté à la gestion de motifs (Parsaye (1999); Calders et al. (2006)).

Dans Zaki et al. (2005), un cadre générique permettant de spécifier des structures de données auxquelles peuvent être associées des fonctions de gestion de motifs est aussi proposé. Tuzhilin et Liu (2002) ont défini des opérateurs de type SQL permettant d'explorer des bases de règles d'association. Dans ces deux cas, bien que des efforts aient été réalisés afin de stocker efficacement des motifs, les langages proposés pour les manipuler sont relativement pauvres. Finalement, dans le domaine de la gestion de bases de motifs, on peut citer également le projet PMML de Grossman et al. (1999) qui permet d'assurer une interopérabilité entre les bases de motifs, en définissant un cadre XML autour du concept de motif. Toutefois, ce cadre s'intéresse plus à la définition d'une représentation structurée des motifs qu'à leur gestion en elle même.

Nos travaux appartiennent également à cette seconde approche basée sur le post-traitement des motifs. Ainsi, nous souhaitons définir une structure de données et des algorithmes efficaces de gestion de grandes bases de motifs. Nous pensons qu'il peut être intéressant pour les utilisateurs de pouvoir obtenir divers ensembles de motifs, construits par exemple par des sessions successives de fouille sur diverses bases de données, puis de disposer d'outils permettant de

[2] http ://www.cinq-project.org/
[3] http ://dke.cti.gr/panda

les gérer efficacement par la suite. En effet, dans de nombreux cas, du fait de problèmes de confidentialité sur les données ou de droit commerciaux d'utilisation des logiciels de fouille, les utilisateurs n'ont en fait pas accès directement aux données, mais seulement aux motifs extraits.

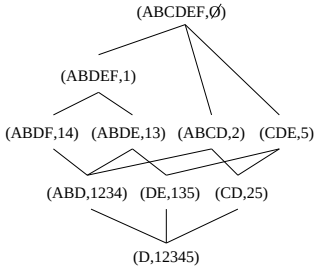
Dans cet article, nous proposons un cadre théorique pour la gestion d'une classe particulière de motifs couramment appelés *concepts* (Wille (1992)). Plus précisément, notre approche se base sur des graphes étiquetés afin de représenter des collections de concepts. Dans ce domaine, peu de travaux ont été réalisés à ce jour. Les recherches les plus proches des nôtres sont sans aucun doute celles de Mielikäinen (2004) qui suggère d'utiliser des automates finis déterministes pour représenter des motifs. Les résultats obtenus expérimentalement montrent que les automates minimaux fournissent une représentation compacte des motifs. Toutefois, Mielikäinen considère des collections d'ensembles d'items et non pas des concepts. Plus encore, il ne propose pas de cadre théorique basé sur des opérateurs algébriques.

La prochaine section rappelle certaines définitions utiles à la compréhension de l'article. Dans la section 3, nous introduisons ensuite la représentation des collections de concepts sous forme de graphe étiqueté et l'algorithme pour les construire. Dans la section 4 nous définissons des opérateurs permettant d'effectuer des requêtes sur ces graphes et pouvant être combinés entre eux en utilisant une algèbre (en un certain sens, cette section est reliée à Diop et al. (2005)).

2 Définitions

Les données D sont définies comme une relation symétrique entre un ensemble d'attributs $\mathcal{A} = \{a_1, a_2, ...\}$ et un ensemble d'objets $\mathcal{O} = \{o_1, o_2, ...\}$. Ces données peuvent être représentées par une matrice booléenne dont les colonnes correspondent aux attributs et les lignes aux objets.

	A	B	C	D	E	F
1	1	1	0	1	1	1
2	1	1	1	1	0	0
3	1	1	0	1	1	0
4	1	1	0	1	0	1
5	0	0	1	1	1	0



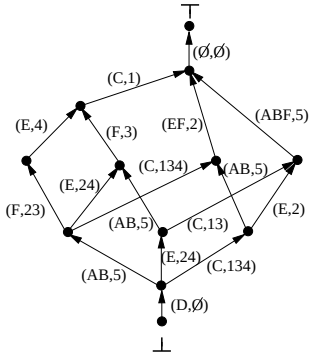


FIG. 1 – *Exemple de données où $\mathcal{A} = \{A, B, C, D, E, F\}$ et $\mathcal{O} = \{1, 2, 3, 4, 5\}$ (gauche), Diagramme de Hasse d'une collection de concepts formels* Concepts(D) *(milieu) et graphe correspondant avec les étiquettes sur les arcs (droite, cf. Sect. 3)*

Une telle base de données peut être, par exemple, le résultat de mesures d'expression de gènes. Dans ce cas, les colonnes représentent les gènes et les lignes correspondent à des situa-

tions biologiques. Il existe une relation entre une situation et un gène si ce gène est sur-exprimé pour cette situation. Dans ce type de bases de données, les biologistes se sont tout particulièrement intéressés à la recherche de concepts formels (Besson et al. (2005)).

Dans la suite, nous utilisons les définitions de Besson et al. (2005). Un **bi-set** est un couple (X, Y) où $X \subseteq \mathcal{A}$ et $Y \subseteq \mathcal{O}$. Un **1-rectangle** est un bi-set (X, Y) tel que tous les attributs de X sont en relation avec tous les objets de Y. Ainsi, dans la matrice booléenne, un 1-rectangle correspond à une sous-matrice contenant uniquement des 1.

Exemple 1 *Dans l'exemple de la Figure 1,* $(ABD, 123)$ *(nous utilisons cette notation simplifiée pour* $(\{A, B, D\}, \{1, 2, 3\})$*) et* $(E, 135)$ *sont des 1-rectangles.* $(ABC, 23)$ *est un bi-set mais, ce n'est pas un 1-rectangle puisque* C *n'est pas en relation avec* 3*.*

La relation d'inclusion $\subseteq$ est définie sur les bi-sets de la façon suivante : $(X_1, Y_1) \subseteq (X_2, Y_2)$ ssi $X_1 \subseteq X_2$ et $Y_1 \subseteq Y_2$. Un **concept formel** est un 1-rectangle maximum pour l'ordre défini sur les bi-sets par l'inclusion. La collection de tous les concepts formels des données D est $\mathsf{Concepts}(D)$ (cf. Fig. 1).

Une relation d'ordre sur les concepts peut alors être définie par : $(X, Y) \preceq (X', Y')$ ssi $X \subseteq X'$ et $Y' \subseteq Y$ (notez le sens de l'inclusion). En munissant la collection de concepts formels de cet ordre, on obtient le treillis des concepts formels. Dans le cas de notre exemple illustratif, ce treillis est représenté dans la Figure 1 (milieu).

3 Représentation d'une collection de concepts

Plusieurs propriétés nous semblent souhaitables pour obtenir une bonne représentation :

- Il doit être possible d'interroger la collection. Par exemple, étant donné une collection de concepts $\mathcal{C}$, il doit être possible de sélectionner tous les concepts qui contiennent un certain attribut ou objet, ou tous les concepts qui contiennent au moins n objets...
- Le résultat d'une requête doit être une collection de concepts utilisant la même représentation que la collection de départ (propriété de fermeture). Ceci est nécessaire pour assurer que l'on puisse enchaîner les requêtes sur une collection de concepts.
- Enfin, la représentation doit respecter la dualité existante entre les ensembles d'attributs et d'objets. Ceci permettra en effet d'obtenir simplement, par exemple, un algorithme pour sélectionner les concepts contenant un objet donné à partir de l'algorithme permettant de sélectionner les concepts contenant un attribut donné.

La sortie des algorithmes d'extraction de concepts (comme D-miner de Besson et al. (2005)) est généralement une liste de concepts. Cela constitue probablement la représentation la plus simple d'une collection de concepts.

Mielikäinen (2004) a proposé d'utiliser un automate pour stocker une collection d'itemsets (un itemset est un ensemble d'attributs). Plusieurs automates sont possibles : par exemple, un arbre de préfixes ou un automate minimal. Cependant, il est nécessaire de définir un ordre sur les attributs pour transformer des ensembles en chaînes de caractères et trouver un tel ordre est très difficile (Mielikäinen (2004)). Cette représentation par automate pourrait être étendue pour représenter une collection de concepts, mais le même problème surgirait : comment transformer une paire d'ensembles en une chaîne de caractères sans utiliser un ordre arbitraire sur les

attributs et les objets ? De plus, manipuler une telle représentation ne semble pas facile. C'est pourquoi nous proposons d'utiliser une autre représentation.

3.1 Notre proposition

Nous proposons d'utiliser un graphe étiqueté fondé sur le diagramme de Hasse de la collection de concepts muni de l'ordre $\preceq$: les sommets du graphe sont les concepts et il existe un arc $X \rightarrow Y$ entre deux concepts X et Y si Y couvre X, i.e., $X \prec Y$ et s'il n'existe pas de concept Z tel que $X \prec Z \prec Y$. Le graphe comporte deux sommets supplémentaires : $\bot$ et $\top$ tels que $(X, Y) \prec \top$ et $\bot \prec (X, Y)$ pour tout (X, Y).

Des étiquettes sont associées soit aux arcs soit aux sommets :

– l'étiquette associée à un sommet (X, Y) est la paire d'ensembles (X, Y) ;

– sur un arc $(X, Y) \rightarrow (X', Y')$, l'étiquette est la paire d'ensembles $(X' \setminus X, Y \setminus Y')$.

Le graphe associé à la collection de tous les concepts des données de l'exemple illustratif est présenté à la Figure 1 (étiquettes sur les arcs).

Une telle représentation présente un double intérêt : premièrement, elle rend l'interrogation de la collection de concepts plus aisée et deuxièmement elle ne nécessite pas, contrairement à d'autres représentations basées notamment sur des automates (Mielikäinen (2004)), de trier les attributs ou des objets selon un ordre difficile à choisir.

3.2 Construction du graphe

Étant donné une liste de concepts extraits par un algorithme tel que D-miner (Besson et al. (2005)), le principe de construction du graphe consiste, à partir d'un graphe contenant seulement les sommets $\top$ et $\bot$, à insérer les concepts l'un après l'autre. Dans le but de simplifier la complexité de l'algorithme, nous proposons d'ajouter les concepts (X, Y) par ordre croissant de taille de X.

Du fait de cet ordre d'insertion des concepts, lorsqu'un nouveau concept $C = (X, Y)$ est inséré, il n'existe pas de concept C' dans le graphe tel que $C \preceq C'$. Ainsi, le seul successeur de C est $\top$ et un arc $C \rightarrow \top$ est ajouté. Il faut ensuite trouver tous les prédécesseurs C' de C dans le graphe (i.e., les concepts C' dans le graphe tels que C couvre C') pour créer les arcs $C' \rightarrow C$. Ceci peut se faire par un parcours en profondeur du graphe.

Finalement, si C couvre un concept C' qui est couvert par $\top$, l'arc $C' \rightarrow \top$ doit aussi être supprimé (puisque $\top$ ne couvre plus C').

4 Les requêtes

Dans cette section, nous étudions différentes opérations réalisables sur une collection de concepts. Deux types de requêtes peuvent être distinguées : les requêtes de sélection d'une part, et les requêtes de projection d'autre part.

4.1 Les requêtes de sélection

Étant donnée une collection $\mathcal{C}$ et un prédicat p, l'opération de sélection suivant p peut être définie par : $\sigma_p(C) = \{(X, Y) \in C \mid p(X, Y) \text{ est vrai}\}$.

Exemple 2 *Parmi les prédicats courants de sélection, on peut citer les exemples suivants (Soulet et Crémilleux (2005)) :*
- *longueur minimale (ou maximale) :* $p(X,Y) = (|X| > \gamma)$ *;*
- *fréquence minimale (ou maximale) :* $p(X,Y) = (|Y| > \gamma)$ *;*
- *surface minimale (ou maximale) :* $p(X,Y) = (|X|.|Y| > \gamma)$ *;*
- *présence (ou absence) d'un attribut (ou d'un objet) dans un concept :* $p(X,Y) = (A \in X)$ *;*
- ...

4.2 Requête de projection

Lors de l'analyse de données d'expression de gènes, les biologistes ont souvent besoin de se restreindre à certains gènes ; ce qui revient à se focaliser sur un sous-ensemble de gènes, par exemple A, B et C.

Pour cela, la solution la plus simple pourrait consister à extraire les concepts non pas sur les données initiales, mais sur une partie contenant seulement les colonnes A, B et C, i.e., sur une projection de la base de données initiale (cf. Fig. 2, droite). Si A est un ensemble d'attributs, nous notons $\pi_A(D)$ la projection de la base D sur les attributs de A.

Cependant, pour diverses raisons, une telle approche ne peut pas toujours être envisagée. En premier lieu, une nouvelle extraction de concepts réalisée dans la projection de la base de données peut s'avérer coûteuse. En second lieu, dans un certain nombre d'applications, les données initiales peuvent ne plus être disponibles, que ce soit pour des raisons de coût de stockage, de perte de données ou encore de confidentialité. Dans de telles situations, si on a toujours accès à la collection de concepts extraits de cette base, on peut se demander s'il ne serait pas possible de retrouver à partir de cette collection de concepts extraits sur toute la base de donnée, la collection de concepts que l'on aurait pu obtenir à partir de la projection de la base de donnée initiale. Répondre à cette question revient en fait à trouver l'opération correspondant à la flèche en pointillés de la Figure 2.

En d'autres termes, nous souhaitons pouvoir calculer $\mathsf{Concepts}(\pi_A(D))$ à partir de $\mathsf{Concepts}(D)$ sans avoir à effectuer une nouvelle extraction dans $\pi_A(D)$. Nous allons démontrer que ceci est en fait réalisable. Pour ce faire, il convient de définir une relation de A-équivalence sur les concepts.

Définition 1 (A**-équivalence)** *Étant donné un ensemble d'attributs* A*, deux concepts* (X,Y) *et* (X',Y') *sont* A*-équivalents ssi* $X \cap A = X' \cap A$*.*

Il s'agit, de manière évidente, d'une relation d'équivalence. La Figure 2 donne un exemple de classes d'équivalence. Ensuite, nous introduisons la proposition suivante :

Proposition 1 *Les classes de* A*-équivalence ont un plus petit élément (pour* $\preceq$*).*

Pour prouver cette proposition, nous utilisons le résultat connu suivant : si $C_1 = (X_1, Y_1)$ et $C_2 = (X_2, Y_2)$ sont deux concepts, alors il existe un concept $C = (X_1 \cap X_2, Y)$ tel que $Y_1 \cup Y_2 \subseteq Y$.

Preuve 1 *Étant donné deux concepts* A*-équivalents minimaux* $C_1 = (X_1, Y_1)$ *et* $C_2 = (X_2, Y_2)$*, alors il existe un troisième concept* $C = (X_1 \cap X_2, Y)$ *tel que* $Y_1 \cup Y_2 \subseteq Y$*.*

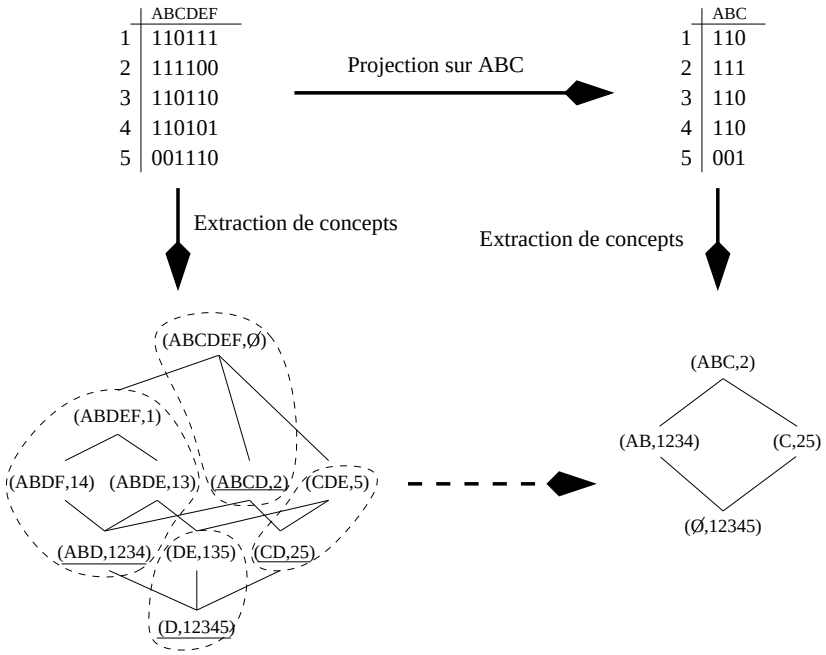


FIG. 2 – *Projection de concepts. Les données de départ D et les concepts correspondants* Concepts(D) *(à gauche) ; La base projeté $\pi_{\{A,B,C\}}(D)$ et $concept(\pi_{\{A,B,C\}}(D))$ (à droite). Les classes de $\{A, B, C\}$-équivalence (en pointillés, cf. Def. 1) et leurs plus petits éléments (soulignés). On peut vérifier que l'intersection de l'ensemble des plus petits éléments avec l'ensemble $\{A, B, C\}$ donne exactement les concepts de $\pi_{\{A,B,C\}}(D)$ (théorème 1).*

De façon évidente, C est A-équivalent à C_1 et C_2 et nous avons aussi $C \preceq C_1$ et $C \preceq C_2$ (par définition de $\preceq$). Par conséquent (comme C_1 et C_2 sont minimaux) $C_1 = C$ et $C_2 = C$ et la classe de A-équivalence de C_1 et C_2 a seulement un élément minimum, i.e., elle a un plus petit élément.

Le théorème suivant permet de caractériser la collection Concepts$(\pi_A(D))$ vis-à-vis de Concepts(D).

Théorème 1 *Étant donné une base de données D et un ensemble d'attributs A, on note PPE_A l'ensemble des plus petits éléments des classes de A-équivalence. Alors*

$$\text{Concepts}(\pi_A(D)) = \{(X \cap A, Y) \mid (X, Y) \in \mathrm{PPE}_A\}.$$

Preuve 2 *Pour prouver ce théorème, nous utilisons le fait que si (X, Y) est un concept dans $\pi_A(D)$ alors il peut être "étendu" pour former un concept (X', Y) dans D où $X' \cap A = X$.*
Première inclusion $\subseteq$:
Soit (X, Y) un concept dans $\pi_A(D)$. (X, Y) peut être étendu en un concept (X', Y) de D. Soit (X'', Y'') un concept A-équivalent à (X', Y) tel que $(X'', Y'') \preceq (X', Y)$. $(X'' \cap A, Y'') = (X, Y'')$ est un 1-rectangle de $\pi_A(D)$. Puisque $Y \subseteq Y''$ et (X, Y) est un concept de $\pi_A(D)$,

Y et Y'' sont égaux. Par conséquent (X'', Y'') est inclus dans (X', Y) et par suite $X'' = X'$ ce qui signifie que $(X'', Y'') = (X', Y)$ et que (X', Y) est le plus petit élément de sa classe de A-équivalence.
Inclusion $\supseteq$:
Soit $(X, Y) \in \mathrm{PPE}_A$. Alors, $(X \cap A, Y)$ est un 1-rectangle dans $\pi_A(D)$. Supposons qu'il existe un 1-rectangle (X', Y') dans $\pi_A(D)$ tel que $(X', Y') \supseteq (X \cap A, Y)$. Alors, $X' = X \cap A$ [sinon $(X \cup X', Y)$ serait un 1-rectangle contenant strictement (X, Y) et par conséquent (X, Y) ne pourrait pas être un concept]. Nous pouvons étendre $(X', Y') = (X \cap A, Y')$ en un concept (X'', Y') de D. Ainsi, $X'' \subseteq X$ [sinon $(X \cup X'', Y)$ serait un 1-rectangle contenant (X, Y) et par suite (X, Y) ne pourrait pas être un concept]. Par conséquent $(X'', Y') \preceq (X, Y)$ et ces deux concepts sont A-équivalents. Donc, ils sont égaux (puisque (X, Y) est un plus petit élément) et $(X \cap A, Y)$ est un 1-rectangle maximal dans $\pi_A(D)$ (pour $\subseteq$), i.e., un concept de $\pi_A(D)$.

Plus généralement, nous pouvons définir une opération de projection sur une collection de concepts de la façon suivante :

Définition 2 (Projection sur une collection de concepts) *Étant donnée une collection de concepts $\mathcal{C}$ dans une base de données D et un ensemble d'attributs A, nous définissons la projection de la collection $\mathcal{C}$ sur A par : $\pi_A(\mathcal{C}) = \{(X \cap A, Y) \mid (X, Y) \in \mathcal{C} \cap \mathrm{PPE}_A\}$ où PPE_A est défini dans le théorème 1.*

Le théorème 1 peut donc se récrire $\mathsf{Concepts}(\pi_A(D)) = \pi_A(\mathsf{Concepts}(D))$. Dans cette égalité, le premier terme π_A désigne la projection de la base de données tandis que le second désigne la projection de la collection de concepts.

Cela signifie que les concepts de la base de données projetée $\pi_A(D)$ peuvent être obtenus par projection des concepts de la base de données initiale D.

4.3 Algèbre

Dans cette section, nous étudions la composition des opérations de projection et de sélection qui ont été définies précédemment sur une collection de concepts.

Il s'agit en fait de trouver une opération permettant de fermer le graphique de la Figure 3. Une solution naturelle est la projection définie précédemment.

En effet, le théorème suivant prouve que le graphique peut être complété à l'aide de cette opération :

Théorème 2 *Étant donnée une collection de concepts $\mathcal{C}$ dans une base de données D, un ensemble d'attributs A et, un prédicat de sélection p tel que pour tout concept (X, Y), $p(X \cap A, Y) = p(X, Y)$, alors $\pi_A \circ \sigma_p(\mathcal{C}) = \sigma_p \circ \pi_A(\mathcal{C})$.*

Preuve 3 *$(X, Y) \in \pi_A(\sigma_p(\mathcal{C})) \Longleftrightarrow \exists (X', Y) \in \sigma_p(\mathcal{C}) \cap \mathrm{PPE}_A$ tel que $X = X' \cap A$ (d'après Def. 2) $\Longleftrightarrow \exists (X', Y) \in \mathrm{PPE}_A \cap \mathcal{C}$ tel que $p(X', Y)$ est vrai et $X = X' \cap A$ $\Longleftrightarrow \exists (X', Y) \in \mathrm{PPE}_A \cap \mathcal{C}$ tel que $p(X, Y)$ est vrai et $X = X' \cap A$ [puisque $p(X', Y) = p(X' \cap A, Y) = p(X, Y)$] $\Longleftrightarrow (X, Y) \in \pi_A(\mathcal{C})$ et $p(X, Y)$ est vrai (d'après Def. 2) $\Longleftrightarrow$ $(X, Y) \in \sigma_p(\pi_A(\mathcal{C}))$*

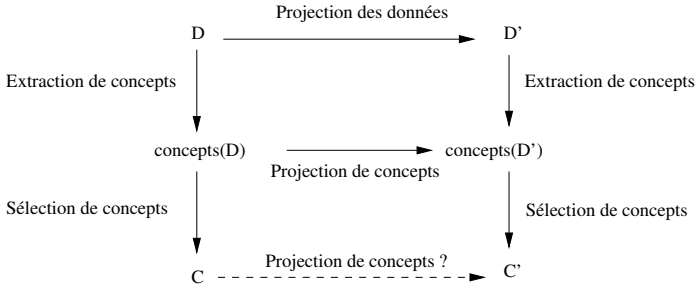


FIG. 3 – *Selection et projection*

La condition sur p peut paraître très forte mais elle est nécessaire. Pour pouvoir réaliser la sélection après la projection, il ne faut pas que la projection restreigne trop l'information contenue dans la collection. Par exemple, si la sélection est définie par $p(X, Y) = (D \in X)$ (i.e., sélection de tous les concepts contenant l'attribut D), alors la sélection ne commute pas avec la projection $\pi_{\{A,B,C\}}$. En effet, après la projection, l'information selon laquelle un concept contient l'attribut D n'est plus disponible. De même dans le modèle relationnel, les opérations de sélection et de projection définies sur des tables relationnelles sont généralement commutatives. Cependant, si la sélection utilise un attribut qui est supprimé par la projection, les deux opérations ne commutent plus.

4.4 Dualité

Dans les deux sections précédentes, nous avons défini la projection d'une collection de concepts sur un ensemble d'attributs A. De manière duale, nous pouvons définir la projection sur un ensemble d'objets O. La relation d'équivalence duale de la A-équivalence (Def. 1) peut être définie de la façon suivante : deux concepts (X, Y) et (X', Y') sont O-équivalents si et seulement si $Y \cap O = Y' \cap O$. Il s'ensuit les résultats duaux correspondants aux théorèmes 1 et 2.

4.5 Algorithmes

Dans cette section, nous proposons les algorithmes permettant de mettre en oeuvre les opérations de projection et de sélection sur le graphe représentant une collection de concepts.

Pour réaliser la projection sur une collection de concepts $\mathcal{C}$ par rapport à un ensemble d'attributs A, il faut pouvoir vérifier qu'un concept est minimal dans sa classe de A-équivalence. Cependant, ceci n'est pas toujours possible sans information complémentaire : il se peut en effet que la collection $\mathcal{C}$ ne contienne pas tous les concepts appartenant à une classe d'équivalence. Dans ce cas, nous pourrions trouver un concept minimum dans cette classe d'équivalence dans $\mathcal{C}$ qui ne soit pas le plus petit élément de cette classe d'équivalence dans $\mathsf{Concepts}(D)$.

Par exemple, supposons que $\mathcal{C}$ contienne tous les concepts de la figure 2 (avant projection) excepté le concept $(D, 12345)$. Alors, si nous effectuons une projection de cette collection sur $\{A, B, C\}$, la projection de $(DE, 135)$ est $(\emptyset, 135)$ qui n'est pas un concept dans $\pi_{\{A,B,C\}}(D)$.

Algorithm 1: projection

Input: Un graphe G représentant une collection $\mathcal{C}$ de concepts et un ensemble A d'attributs
Output: Un graphe G' représentant la collection $\pi_A(\mathcal{C})$
forall $X \in \mathcal{C}$, X *non marqué* **do**
  PPE[X]= X
forall $X \in \mathcal{C}$, X *non marqué, dans l'ordre topologique* **do**
  if *PPE[X]* = X **then** // X est peut être dans PPE_A
    if $\exists Y \in predecesseur(X)$, Y *marqué et classe(Y)* = *classe(X)* **then** // X n'est pas dans PPE_A
      PPE[X] = NULL
    else // X est dans PPE_A
      ajoute_sommet(proj(X), G')
      forall $X' \in prdcesseur(X)$, X' *marqué* **do**
        ajouter_sommet_marqué(proj(X'), G')
        ajoute_arc($proj(X') \rightarrow proj(X)$, G')
  forall $Y \in predecesseur(X)$, Y *non marqué* **do**
    if *classe(Y)* = *classe(X)* **then**
      PPE[Y] = PPE[X]
forall *arc* $X \rightarrow Y$ *dans* G, X *et* Y *non marqués* **do**
  ajoute_arc(proj(PPE[X]) $\rightarrow$ proj(PPE[Y]), G')

Ceci est dû au fait que $(DE, 135)$ n'est pas le plus petit élément de sa classe d'équivalence ce qui n'est pas détectable sans information additionnelle.

C'est la raison pour laquelle il est nécessaire d'ajouter des informations dans le graphe de notre représentation. Étant donnée une collection de concepts $\mathcal{C}$, nous ajoutons au graphe les concepts qui sont à la frontière de la collection. Par concepts frontières, nous entendons les concepts qui sont soit successeurs soit prédécesseurs d'un concept appartenant à la collection. Ces concepts additionnels portent une marque spécifique et ne sont pas reliés aux sommets $\top$ et $\bot$ (ils sont insérés dans le graphe au moyen de la fonction `ajouter_sommet_marqué`); ils sont seulement reliés au(x) concept(s) de la collection dont ils sont successeurs ou prédécesseurs. Bien évidemment, lors des opérations de sélection ou de projection, cette information additionnelle sera conservée.

La projection peut être réalisée en suivant l'Algorithme 1. Pour tout sommet X, l'algorithme calcule `PPE[X]` qui est le plus petit élément de la classe de A-équivalence de X. Si ce plus petit élément n'est pas dans la collection, alors `PPE[X]=NULL`. Les plus petits éléments des classes d'équivalence sont insérés dans le nouveau graphe G' et les arcs sont ajoutés à G'. Dans cet algorithme, $(X \cap A, Y)$ est noté `proj(X,Y)`.

Dans le cas général, pour effectuer une sélection de la collection de concepts selon un prédicat p, il est nécessaire de parcourir le graphe représentant la collection et de tester le prédicat p pour chacun des concepts. Cependant le parcours peut être réduit lorsque p est monotone ou anti-monotone. Rappelons qu'un prédicat p est anti-monotone ssi $(\neg p(X,Y) \wedge ((X,Y) \preceq (X',Y'))) \Rightarrow \neg p(X',Y')$ et monotone ssi $(\neg p(X,Y) \wedge ((X',Y') \preceq (X,Y))) \Rightarrow \neg p(X',Y')$. Dès lors, si p est anti-monotone, une exploration *bottom up* (de $\bot$ vers $\top$) peut être réalisée et dès qu'un concept X ne vérifiant pas p est trouvé, il n'est pas nécessaire d'explorer ses successeurs (cf. Alg. 2). De façon équivalente, pour un prédicat monotone, une exploration *top down* permettra de parcourir partiellement le graphe.

Algorithm 2: selection_AM

Input: Un graphe G représentant une collection $\mathcal{C}$ de concepts et un prédicat de sélection anti-monotone p
Output: Un graphe G' représentant la collection $\sigma_p(\mathcal{C})$
$G =$ graphe_vide
`ajoute_sommet`($\top, G$)
`ajoute_sommet`($\bot, G$)
$E = \emptyset$ `// E est une variable globale`
explore($\bot$)
return G'

Procedure `explore`(*sommet* V)

$E = E \cup \{V\}$
forall $X \in$ *prédécesseur*(V) *et* X *marqué* **do**
  `ajouter_sommet_marqué`(X, G')
  `ajoute_arc`($X \rightarrow V, G'$)
link_to_top = vrai
forall $X \in$ *successeur*(V) **do**
  if $p(X)$ *et* X *non marqué* **then**
    link_to_top = faux
    if $X \notin E$ **then**
      `ajoute_sommet`(X, G')
      explore(X)
    `ajoute_arc`($V \rightarrow X, G'$)
  else
    `ajouter_sommet_marqué`(X,G')
    `ajoute_arc`($V \rightarrow X, G'$)
if *link_to_top* **then**
  `ajoute_arc`($V \rightarrow \top$)

5 Conclusion

Dans cet article, nous présentons un cadre théorique pour la représentation et l'interrogation d'une collection de concepts. Nous proposons de stocker la collection sous forme de graphe et nous définissons deux types d'opérations applicables sur le graphe : la sélection et la projection. Nous envisageons à présent de poursuivre ce travail dans plusieurs directions :

Tout d'abord il serait intéressant d'étudier le passage à l'échelle de cette représentation sur des données réelles et de la comparer avec d'autres types de représentations, comme par exemple les automates. En particulier, l'étude des relations entre la taille de la représentation et les caractéristiques des données à partir desquelles elle a été générée serait intéressante.

Ensuite, notre représentation sous forme de graphe paraît avantageuse pour l'interrogation d'une collection mais en revanche, elle n'est pas très compacte. Aussi, il pourrait être intéressant de prévoir deux modèles de représentation : le premier, très compact, destiné au stockage (sur disque) , le second sous forme de graphe utilisable pour l'interrogation.

Finalement, un certain nombre de recherches ont été consacrées à la généralisation des concepts et aux clusters de concepts. La définition d'une opération de regroupement sur le graphe supportant ces différentes généralisations mériterait également d'être étudiée.

Références

Besson, J., C. Robardet, J.-F. Boulicaut, et S. Rome (2005). Constraint-based concept mining and its application to microarray data analysis. *IDA 9*(1), 59–82.

Boulicaut, J.-F., M. Klemettinen, et H. Mannila (1999). Modeling KDD processes within the inductive database framework. In *First International Conference on Data Warehousing and Knowledge Discovery*, Volume 1676 of *LNCS*, pp. 293–302.

Boulicaut, J. F. et C. Masson (2005). Data mining query languages. In *The Data Mining and Knowledge Discovery Handbook*, pp. 715–727. Springer.

Calders, T., B. Goethals, et A. Prado (2006). Integrating pattern mining in relational databases. In *PKDD*, Volume 4213 of *LNCS*, pp. 454–461.

Catania, B. et A. Maddalena (2006). *Pattern Management : Practice and Challenges*, pp. 280–317. Processing and Managing Complex Data for Decision Support. Idea Group Publishing.

Catania, B., A. Maddalena, M. Mazza, E. Bertino, et S. Rizzi (2004). A framework for data mining pattern management. In *PKDD*, Volume 3202 of *LNCS*, pp. 87–98.

Diop, C. T., A. Giacometti, D. Laurent, et N. Spyratos (2005). Computation of mining queries : An algebraic approach. In *Constraint-Based Mining and Inductive Databases*, Volume 3848 of *LNCS*, pp. 102–126.

Grossman, R. L., S. Bailey, A. Ramu, B. Malhi, P. Hallstrom, I. Pulleyn, et X. Qin (1999). The management and mining of multiple predictive models using the predictive model markup language (PMML). In *Information and Software Technology*, Volume 41, pp. 589–595.

Imielinski, T. et H. Mannila (1996). A database perspective on knowledge discovery. *Comm. ACM 39*(11), 58–64.

Mielikäinen, T. (2004). An automata approach to pattern collections. In *KDID*, Volume 3377 of *LNCS*, pp. 130–149.

Panda. Patterns for next-generation database systems (2001-2004). FET/IST-2001-33058.

Parsaye, K. (1999). From datamagement to pattern management. *DM Rev. Mag.*.

Raedt, L. D. (2002). A perspective on inductive databases. *SIGKDD Explorations 4*(2), 69–77.

Raedt, L. D., M. Jaeger, S. Lee, et H. Mannila (2002). A theory of inductive query answering. In *Proc. ICDM*, pp. 123–130.

Soulet, A. et B. Crémilleux (2005). An efficient framework for mining flexible constraints. In *PAKDD*, Volume 3518 of *LNCS*, pp. 661–671.

Tuzhilin, A. et B. Liu (2002). Querying multiple sets of discovered rules. In *Proceedings ACM SIGKDD*, pp. 52–60. ACM.

Wille, R. (1992). Concept lattices and conceptual knowledge systems. *Comp. math. applied 23*(6-9), 493–515.

Zaki, M. J., N. Parimi, N. De, F. Gao, B. Phoophakdee, J. Urban, V. Chaoji, M. A. Hasan, et S. Salem (2005). Towards generic pattern mining. In *Formal Concept Analysis*, pp. 1–20.

Summary

Data mining algorithms are now able to efficiently deal with huge amount of data. Nevertheless, users are often overwhelmed by the large quantity of patterns extracted in such a situation. Moreover, some privacy issues, or some commercial one may prevent the users from mining the data by themselves. Thus, the users may not have the possibility to perform many experiments integrating various constraints in order to focus on specific patterns they would like to extract. Post processing of patterns may be an answer to that drawback. Thus, in this paper we present a framework that could allow end users to manage collections of patterns. We propose to use an efficient data structure on which some algebraic operators may be used in order to retrieve or access patterns in pattern bases.